CSE 333 – SECTION 5

C++ Classes, Const and References;

Some slides referenced from CSE 333 -Winter 2018 slides
Logistics

Friday (tomorrow)
   Exercise 6 @ 10:00 am

Friday (1 week from now):
   HW2 @ 11:00 pm

Mid-quarter Survey
Section Plan

- C++ const/reference refresher
- References Problem
- C++ Classes
- Mult-Choice Problem
- STL
Example

- Consider the following code:

```c
int x = 5;
int &refx = x;
int *ptrx = &x;
```

Similar in syntax to the `*` in pointer declarations

What are some tradeoffs to using pointers vs references?

Legend

Red Thing = “can’t change the box it’s next to”
Black = “writeable/readable”
## Summary

**Pointers vs. References:**

<table>
<thead>
<tr>
<th>Pointers</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can move to different data via reassignment/pointer arithmetic</td>
<td>References the same data for its entire lifetime</td>
</tr>
<tr>
<td>Can be initialized to NULL</td>
<td>No sensible “default reference”</td>
</tr>
<tr>
<td>Used for output parameters e.g. MyClass* output</td>
<td>Used for input parameters e.g. const MyClass&amp; input</td>
</tr>
</tbody>
</table>
When would you prefer reference to pointer as function parameters?

- When you don’t want to deal with pointer semantics, use references
- When you don’t want to copy stuff over (doesn’t create a copy, especially for parameters and/or return values), use references
- Style wise, we want to use **references for input parameters** and **pointers for output parameters**, with the output parameters declared last
Const

- Mark a variable with const to make a compile time check that a variable is never reassigned
- Does not change the underlying write-permissions for this variable

```c
int x = 42;

const int* ro_ptr = &x;  // Read only
int* rw_ptr = &x;  // x can still be modified with rw_ptr!
int* const ptr = &x;  // Only ever points to x
```
Example (Ex 1)

- Consider the following code:

```c
int x = 5, y = 0;
int &refx = x;
const int &ro_refx = x;
int *ptry = &y;
const int *ro_ptr1 = &y;
int *const ro_ptr2 = &y;
```

```
5
x, refx, ro_refx
```

Tip: Read the declaration “right-to-left”

Legend

Red Thing = “can’t change the box it’s next to”
Black = “writeable/readable”
Example (Ex 1)

Consider the following code:

```cpp
int x = 5, y = 0;
int &refx = x;
const int &ro_refx = x;
int *ptry = &y;
const int *ro_ptr1 = &y;
int *const ro_ptr2 = &y;
```

Tip: Read the declaration “right-to-left”

Legend
Red Thing = “can’t change the box it’s next to”
Black = “writeable/readable”
Example (Ex 1)

- Consider the following code:

  ```c
  int x = 5, y = 0;
  int &refx = x;
  const int &ro_refx = x;
  int *ptry = &y;
  const int *ro_ptr1 = &y;
  int *const ro_ptr2 = &y;
  ```

Which results in a compiler error?

- **ok** bar(refx);
- **Error** bar(ro_refx);
- **ok** foo(refx);

Legend

- **Red Thing** = “can’t change the box it’s next to”
- **Black** = “writeable/readable”

```c
void foo(const int &arg);
void bar(int &arg);
```
Example (Ex 1)

Consider the following code:

```c
int x = 5, y = 0;
int &refx = x;
const int &ro_refx = x;
int *ptry = &y;
const int *ro_ptr1 = &y;
int *const ro_ptr2 = &y;
```

Which of these results in a compiler error?

- ok  ro_ptr1 = (int*)0xDEADBEEF;
- Error ro_ptr2 = ro_ptr2 + 2;
- Error *ro_ptr1 = *ro_ptr1 + 1;

Legend

Red Thing = “can’t change the box it’s next to”
Black = “writeable/readable”
C++ Classes
Class Organization

Point.h

```cpp
class Point {
public:
    Point(int x, int y);
    int get_x() { return x_; }
    int get_y() { return y_; }
    double Distance(Point & p);
    void SetLocation(int x, int y);
private:
    int x_;  // x component of point
    int y_;  // y component of point
};
```

Class declaration goes in Point.h, implementation goes in Point.cc.

Point.cc

```cpp
Point::Point(int x, int y){
    x_ = x;
    this->y_ = y;
}

double Point::Distance(Point & p){
    double xdiff = pow(x_ - p.x_, 2);
    double ydiff = pow(y_ - p.y_, 2);
    return sqrt(xdiff + ydiff);
}

void Point::SetLocation(int x, int y){
    x_ = x;
    this->y_ = y;
}
```
Class .h files

Point.h

class Point {
  public:
    Point(int x, int y);
    int get_x() { return x_; }
    int get_y() { return y_; }
    double Distance(Point & p);
    void SetLocation(int x, int y);
  private:
    int x_; 
    int y_; 
};

- Includes the class declaration.
- Can specify member functions and variables and whether they are public/private
- Can have implementation of functions, usually only done with simple functions (e.g. getters)
Class .cc files

Contains member function definitions. These are indicated by:
Class_Name::Func_name(){

If not specified as part of the class, it cannot access private class members, and probably won't compile.

```cpp
Point::Point(int x, int y){
    x_ = x;
    this->y_ = y;
}

double Point::Distance(Point &p){
    double xdiff = pow(x_ - p.x_, 2);
    double ydiff = pow(y_ - p.y_, 2);
    return sqrt(xdiff + ydiff);
}

void Point::SetLocation(int x, int y){
    x_ = x;
    this->y_ = y;
}
```
What about “const” object methods?
Cannot mutate the object it’s called on. Trying to change x_ or y_ inside will cause a compiler error!
class MultChoice {
public:
    MultChoice(int q, char resp) : q_(q), resp_(resp) {} // 2-arg ctor
    int get_q() const { return q_; }
    char get_resp() { return resp_; }
    bool Compare(MultChoice &mc) const; // do these MultChoice's match?

private:
    int q_; // question number
    char resp_; // response: 'A','B','C','D', or 'E'
}; // class MultChoice

<table>
<thead>
<tr>
<th>Code</th>
<th>Error?</th>
<th>Code</th>
<th>Error?</th>
</tr>
</thead>
<tbody>
<tr>
<td>int z = 5;</td>
<td></td>
<td>int z = 5;</td>
<td></td>
</tr>
<tr>
<td>const int *x = &amp;z;</td>
<td></td>
<td>int *const w = &amp;z;</td>
<td></td>
</tr>
<tr>
<td>int *y = &amp;z;</td>
<td></td>
<td>const int *const v = &amp;z;</td>
<td></td>
</tr>
<tr>
<td>x = y;</td>
<td></td>
<td>*v = *w;</td>
<td></td>
</tr>
<tr>
<td>*x = *y;</td>
<td></td>
<td>*w = *v;</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 3

class MultChoice {
    public:
        MultChoice(int q, char resp) : q_(q), resp_(resp) {} // 2-arg ctor
        int get_q() const { return q_; }
        char getResp() { return resp_; }
        bool Compare(MultChoice &mc) const; // do these MultChoice's match?

    private:
        int q_; // question number
        char resp_; // response: 'A', 'B', 'C', 'D', or 'E'
}; // class MultChoice

<table>
<thead>
<tr>
<th>Code</th>
<th>Error?</th>
<th>Code</th>
<th>Error?</th>
</tr>
</thead>
<tbody>
<tr>
<td>int z = 5;</td>
<td>y</td>
<td>int z = 5;</td>
<td>y</td>
</tr>
<tr>
<td>const int *x = &amp;z;</td>
<td>y</td>
<td>int *const w = &amp;z;</td>
<td>y</td>
</tr>
<tr>
<td>int *y = &amp;z;</td>
<td>y</td>
<td>const int *const v =</td>
<td>y</td>
</tr>
<tr>
<td>x = y;</td>
<td>y</td>
<td>&amp;z;</td>
<td>y</td>
</tr>
<tr>
<td>*x = *y;</td>
<td>y</td>
<td>*v = *w;</td>
<td>y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*w = *v;</td>
<td>y</td>
</tr>
</tbody>
</table>
class MultChoice {
    public:
        MultChoice(int q, char resp) : q_(q), resp_(resp) { } // 2-arg ctor
        int get_q() const { return q_; }
        char get_resp() { return resp_; }
        bool Compare(MultChoice &mc) const; // do these MultChoice's match?
    private:
        int q_;  // question number
        char resp_; // response: 'A','B','C','D', or 'E'
}; // class MultChoice

<table>
<thead>
<tr>
<th>Code</th>
<th>Error?</th>
<th>Code</th>
<th>Error?</th>
</tr>
</thead>
<tbody>
<tr>
<td>const MultChoice m1(1,'A');</td>
<td></td>
<td>const MultChoice m1(1,'A');</td>
<td></td>
</tr>
<tr>
<td>MultChoice m2(2,'B');</td>
<td></td>
<td>MultChoice m2(2,'B');</td>
<td></td>
</tr>
<tr>
<td>cout &lt;&lt; m1.get_resp();</td>
<td></td>
<td>cout &lt;&lt; m1.getResp();</td>
<td></td>
</tr>
<tr>
<td>cout &lt;&lt; m2.get_q();</td>
<td></td>
<td>m1.Compare(m2);</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>m2.Compare(m1);</td>
<td></td>
</tr>
</tbody>
</table>
Exercise 3

```cpp
class MultChoice {
public:
    MultChoice(int q, char resp) : q_(q), resp_(resp) { } // 2-arg ctor
    int get_q() const { return q_; }
    char get_resp() { return resp_; }
    bool Compare(MultChoice &mc) const; // do these MultChoice's match?

private:
    int q_; // question number
    char resp_; // response: 'A', 'B', 'C', 'D', or 'E'
}; // class MultChoice
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Error?</th>
<th>Code</th>
<th>Error?</th>
</tr>
</thead>
<tbody>
<tr>
<td>const MultChoice m1(1,'A');</td>
<td>N</td>
<td>const MultChoice m1(1,'A');</td>
<td>N</td>
</tr>
<tr>
<td>MultChoice m2(2,'B');</td>
<td>N</td>
<td>MultChoice m2(2,'B');</td>
<td>N</td>
</tr>
<tr>
<td>cout &lt;&lt; m1.get_resp();</td>
<td>Y</td>
<td>m1.Compare(m2);</td>
<td>N</td>
</tr>
<tr>
<td>cout &lt;&lt; m2.get_q();</td>
<td>N</td>
<td>m2.Compare(m1);</td>
<td>Y</td>
</tr>
</tbody>
</table>
What would you change about the class declaration to make it better?

class MultChoice {
public:
    MultChoice(int q, char resp) : q_(q), resp_(resp) { } // 2-arg ctor
    int get_q() const { return q_; }
    char get_resp() { return resp_; }
    bool Compare(MultChoice &mc) const; // do these MultChoice's match?

private:
    int q_;    // question number
    char resp_; // response: 'A','B','C','D', or 'E'
}; // class MultChoice
C++ STL
Templates

- C++ supports templates to facilitate generic data types
  - Parametric polymorphism - similar to Java generics but different in details (mainly implementation)
  - Example:
    - `vector<int> x`  \text{vector of ints}
    - `vector<string> x`  \text{vector of strings}
    - `vector<vector<float>> x`  \text{vector of (vectors of floats)}
STL (Standard Template Library)

- Set of C++ template classes that provide common programming functionality
  - A string class
  - Generic containers: queue, list, stack, vector, bitset, associative array, deque, and set
    - Iterators
    - Algorithms
  - And much more...
STL vector

- A generic, dynamically resizable array
- Elements are stored in contiguous memory locations
  - Elements can be accessed using pointer arithmetic if you’d like
  - Random access is $O(1)$ time
- Adding/removing from the end is cheap (amortized constant time)
- Inserting/deleting from the middle or start is expensive (linear time)
STL iterator

- Each container has an iterator class
- Ranges from **begin** to **end**
  - [begin, end)